DOCUMENT RESUME

ED 444 452 IR 020 089

AUTHOR Curtis, Ronald; Najarian, John P.
TITLE Active Learning Strategies in Computer Graphics. Research Paper: Connecting Technology to Teaching and Learning.
PUB DATE 2000-00-00
NOTE 10p.; Connecting @ the Crossroads, NECC 2000: National Educational Computing Conference Proceedings (Atlanta, GA, June 26-28, 2000); see IR 020 086.
PUB TYPE Opinion Papers (120) -- Speeches/Meeting Papers (150)
EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Active Learning; *Computer Assisted Instruction; *Computer Graphics; Computer Science Education; Higher Education; Instructional Innovation; Learner Controlled Instruction; Learning Activities; Student Participation; Visual Stimuli

ABSTRACT

When competing with mass media and other forms of information delivery, a verbal lecture does not have enough sensory impression. What students read in a passive framework is barely remembered, less so in pre-examination cramming. Weaker students have difficulties in reading, most notably with abstract and mathematical concepts. These problems are magnified in computer graphics, where mathematical expressions/structures and programming projects may prove too formidable. In this paper, active learning models are applied to computer graphics. Lectures are augmented in real time with student activities, inquiry-based reasoning, and other methods of initiating student contribution to the learning experience. This approach promotes a more comprehensive, deeper, and more memorable understanding of the theory, principles, and methodologies of computer graphics. A progression of such activities is presented as they are applied in the course. Six figures illustrate the text. Contains 14 references. (AEF)

Reproductions supplied by EDRS are the best that can be made
from the original document.

ED 444 452

Research Paper: Connecting Technology to Teaching and Learning

Active Learning Strategies in Computer Graphics

Ronald Curtis
Department of Computer Science
William Paterson University of New Jersey
Wayne, NJ 07470
curtisr@cs.wpunj.edu

John P. Najarian
Department of Computer Science
William Paterson University of New Jersey
Wayne, NJ 07470
najarian@cs.wpunj.edu

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL HAS BEEN GRANTED BY

D. Ingham

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

1

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

- This document has been reproduced as received from the person or organization originating it.
- Minor changes have been made to improve reproduction quality.
- Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

Key Words: active learning, aesthetics, OpenGL®, ray tracing

Abstract

In this paper, active learning models are applied to computer graphics. Lectures are augmented in real time with student activities, inquiry-based reasoning, and other methods of initiating student contribution to the learning experience. This approach promotes a more comprehensive, deeper, and more memorable understanding of the theory, principles, and methodologies of computer graphics. A progression of such activities is presented as they are applied in the course.

Introduction

The classical model of lecture-homework-exam does not meet the challenge of contemporary students. When competing with mass media and other forms of information delivery, a verbal lecture does not leave enough sensory impression. Likewise, even what students read in a passive framework is barely remembered, less so in pre-examination cramming. Weaker students have difficulties in reading, most notably with abstract and mathematical concepts. These problems are magnified in computer graphics, where mathematical expressions/structures and programming projects may prove too formidable.

The solution to these problems requires a radically different approach to lectures, homework, and projects. Lectures should involve periodic breaks in which students perform activities that reinforce the concepts being inculcated. The activities should be simple at first and based upon student experience or a familiar context. In the beginning, this may mean giving the students programs with which to experiment rather than expecting that they write the code.

IR020089

BEST COPY AVAILABLE

Foundations and Aesthetics

At the earliest stages, students need to understand the concepts of color and resolution, especially appreciating their effect on the images generated. To achieve that, a useful exercise is to have students download sample GIF or JPEG images and view them in a standard imaging package such as Photoshop® or LView®. There are many such packages in the public domain. Next, have the students magnify the images to see the inherent pixel nature of the image. Another useful exercise is to have them view the photograph at 4, 16, 256, ~65,000, and 16 million colors. They quickly recognize the associated information content. Have them save the files at each step. They directly see the "space complexity" measure by looking at the file sizes. To demonstrate the effects of compression, have students save an image as a bitmap (such as uncompressed TIFF).

These activities also prove useful when preceding a lecture on the hardware and history of computer graphics. Most students have little idea that the era of mainframe and minicomputer dominance involved little graphics beyond character-mode printed posters. Some may remember low-resolution four-color applications of the early 1980s. To best contrast this evolution, have them compare small, simple overview-perspective games with textured-bitmapped first-person three-dimensional games. Restrict this time period to at most five minutes, the goal being discernment, not frivolity. This can be followed by a discussion of the historical trends of graphics hardware during the 1980s and 1990s. The need for increasing memory model size, graphics memory, display resolution, CPU speed, and for coprocessors takes on a greater reality. The importance of the transition from two dimensions to three becomes apparent. Another useful exercise in hardware appreciation is to run several graphical programs (from the student assignments) with specialized graphic-instructional primitives utilizing the coprocessor, and then run those programs without them. A graphical simulation of raster scan operations would be useful but none are available.

To facilitate this initial experience-orientation, a textbook should introduce colorful samples of computer graphics early. Hearn and Baker (1997; the primary text used in this course) and Firebaugh (1993) provide numerous illustrations at this stage.

Graphic Primitive Programming

Programming in computer graphics often begins with designing a line-draw function from a single-pixel dot function. One useful exercise is to hand out graph paper and have students "x"-out boxes to draw lines. They easily accomplish this for drawing y=x and y=0. At y=(3/7)*x and y=(1/4)*x, serious problems become evident. At this point, they are ready to code a simple incremental algorithm (such as digital differential analyzer—DDA). After several runs, have students try their algorithm with y=100*x. Students arrive on their own at the conclusion to switch the looping dimension depending on slope.

Bresenham's line algorithm requires even more preparation. One activity students enjoy is calculating. Using a spreadsheet package, have students compute intermediate numerical values that Bresenham requires and then use the chart facility to view the results. After this prototyping activity, students are generally more confident in proceeding to the program design stage.

Another useful exercise is for students to form teams, each responsible for a distinct algorithm for solving the same problem. In the circle-generating problem, each team has a different algorithm to program and they post their timing results on the board. This is also a good time to compare their results with those produced by a coprocessor's routine.

Two-Dimensional Programming

The next exercise is to design a plotting routine for an arbitrary parametric curve in the plane (see Figures 1 and 2 for examples of parametric curves). A good first step is to have students research on the Web or in the library about particular parametric curves, such as hypotrochoids. After studying the curves and getting formulas for them, students insert those formulas in the plotter routine. This routine (only 25–30 lines of code) generates 1,000 points of that curve, computes the maximum and minimum values, and then scales and plots all points based on those maximum and minimum values.

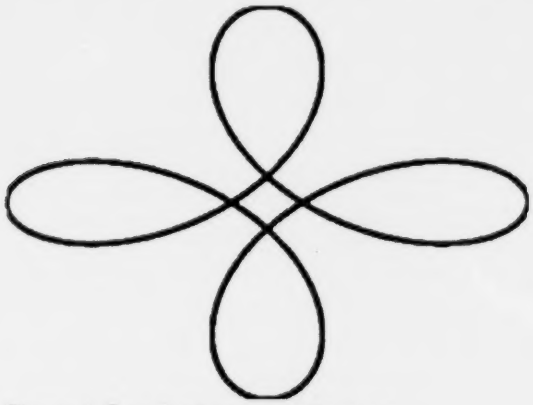

Figure 1. Standard parametric curve.

A bountiful source of several major families of parametric curves is Lawrence (1972). Introducing parametrics is important because of the decline in their coverage in a significant portion of modern calculus texts, despite their central role in several areas of computer graphics. They serve as a useful introduction to multivalued functions, broaden student understanding of piecewise linear graphics in the plane, and have applications in simulation.

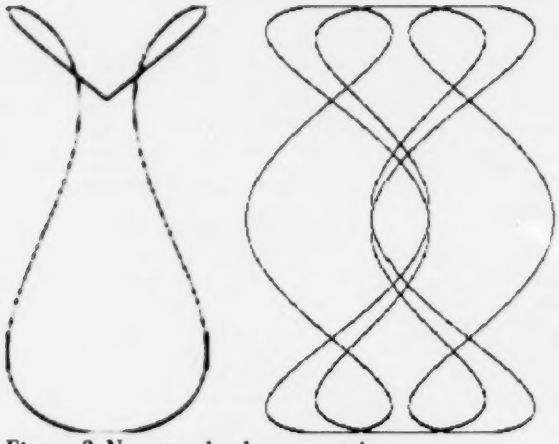

Figure 2. Nonstandard parametric curves.

The team approach works well with fill algorithms. In each team, one person supervises the trial drawings and the key logic therein. The second person converts the samples and reasoning into a program. A third person constructs test inputs; these inputs should adequately address problem situations and thoroughly test

BEST COPY AVAILABLE

the code. Finally, the fourth person is responsible for debugging and testing. The debug phase often produces surprising results, the most amusing being fills that go beyond boundaries and inundate the screen.

Anti-aliasing models can be tested by editing an enlarged image and then reducing it to see the actual results at the normal scale. This is best accomplished with line segments since this activity requires some effort even for small segments; that segment can then be duplicated and concatenated to see the final results.

Transformations are easily experimented with; most image-processing packages have such options. At the stage of students programming transformations, the situation has some interesting side effects. When working with line segments defined by endpoints, the matrix transformations work well. However, when students run their transformation procedures on images, the side effects from rotations and scaling are quite shocking. The need for interpolation becomes acute.

Clipping algorithms can be covered by first providing some examples for the whole class to solve by hand on paper, providing the solutions when errors are encountered, and then forming teams for the actual programming. The problem in programming is that students often "visualize" the solution before applying the algorithms in the class problem-solving efforts. During the programming, no such global intuitions are available during execution. Teams of at least four are needed here; at least one student should concentrate just on the algorithm's steps, not the coding issues, to guarantee adherence to the method assigned. Roles of students in each team should be based on their decisions; to impose a rotation of functions on students often irritates them by blocking their opportunity to contribute in areas of their strengths.

Three-Dimensional Wiring

The introduction to three dimensions requires an extensive review of vector and matrix operations. Intermix lecturing with student calculation exercises (see Figure 3).

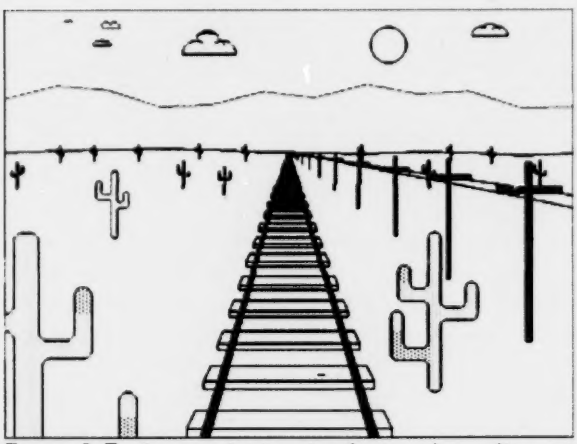

Figure 3. Early exercise in wire-frame, three-dimensional perspective calculation, depth cues, and random-image generation (desert scene).

It is beneficial to have students verify their results on a simple wire-frame modeling program. As they progress from the basics to the three-dimensional transformations, this visual verification becomes indispensable. Students should program short routines to read/construct and display polygon tables and quadrilateral meshes. They can then modify their two-dimensional transformation procedures to three

dimensions (a short activity). At this point, have the students modify their 1,000-point plotter in two dimensions to three dimensions and apply transformations to give perspective. The resultant surface plotter is quite an accomplishment.

Students consistently dread the spline representation. Having students generate sample cubics with their two-dimensional plotter routine helps them understand the versatility of this curve. A formal discussion of curvature and quadratic limitations will give some background but rarely convinces them as well. A computer-aided design (CAD) or drawing package with splines built-in is an effective way for students to visualize the effects of control points. After several proofs and many numerical examples, assign a different spline algorithm to each team. After computing coefficients, have students test their results by running the plotter routine on the spline and also plotting the data points.

Solid Phase: Ray Tracing

Introducing a public domain ray tracer such as a Persistence of Vision™ Ray Tracer (POV-Ray) is an effective way to approach constructive solid geometry (see Figure 4). Students find the programming less pressured, can concentrate on concepts, and produce results too time-consuming to completely program otherwise. This approach was presented in Owen, Larrondo-Petrie, and Laxer (1994) and Schweitzer (1990). POV-Ray includes online interactive tutorials. POV-Ray code samples on the Web are abundant. Students develop a solid appreciation of three-dimensional positioning, perspective, lighting, color, and texture as they construct several scenes. Several books by Waite Group Press are complete introductions to ray tracing and animation with POV-Ray, most notably Young and Wells (1994) and Bowermaster (1994). Unfortunately, all of them are out of print but are worth borrowing through interlibrary loan with their CDs. The C/C++ code can interact with the user and then call POV-Ray routines to accomplish the task. One popular activity is to write recursive C code for a Lindenmayer system. This generates point and arc information, which is then rendered using POV-Ray. This may be a student's best opportunity to see the inherent power of recursion.

Figure 4. POV-Ray generated scene.

Ray tracing prepares students for actual C/C++ programming with depth, shadows, reflections, and other three-dimensional characteristics. Pedagogical exercises with folded paper models can serve as a prelude to algorithm design for binary space-partitioning (BSP) trees and hidden line removal. Bringing a flashlight and metal objects to class will illuminate during a lecture on normals, lighting, and reflection. By turning off the overhead lights and closing the shades, students can get a more dramatic and realistic view than any ray

tracer could produce. This serves as an effective lab exercise. The mathematical model and subsequent C/C++ code follows.

One of the benefits of using C/C++ is that scene generation occurs in real time, so that if a bug occurs, students can predict and observe when the scene went bad. It is also more amusing watching objects and shadows get generated.

Fractals and Randomized Models

The simplest exercise in randomization (Monte Carlo methods) in computer graphics is to have students write a C/C++ program to generate a wire-frame grid and then use a random-number generator to raise or depress grid-points on that surface a random but carefully bounded amount (like wrinkling). This is an easy and short program, described in Pokorny (1994). Going a step further, have students use nonuniform random-number generators to produce different textures. As a final step, apply color variations depending on surface height. Applying colors carefully, one can achieve the effect of geological strata.

Another application of random numbers and colors is in producing cloud formations (see Figure 5). Using a random-number generator with a Gaussian distribution and applying shades of white on a blue background in cluster patterns, students can produce impressive results.

As an independent but not mutually exclusive methodology, fractals and space-filling curve generation also gives students an occasion to write functions in class that generate complex natural objects and scenes (see Figure 6). By putting more parameters in to this function (such as angles and length ratios), broad categories of fractals can be generated.

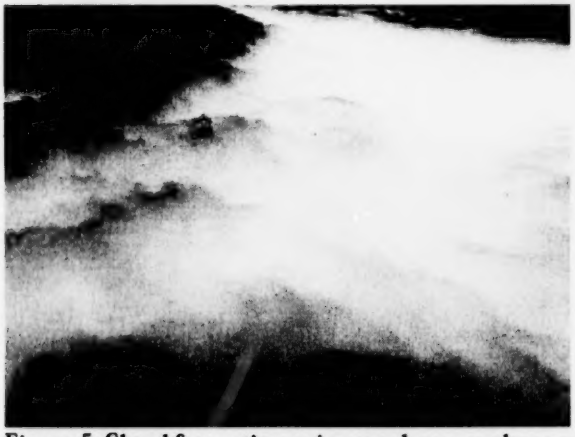

Figure 5. Cloud formation using random-number generators (compare with Figure 6).

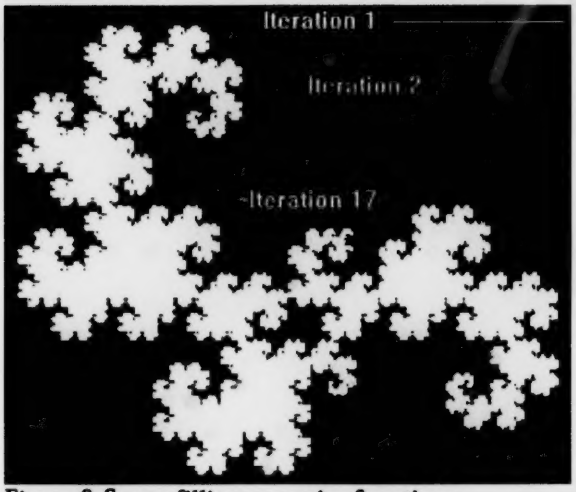

Figure 6. Space-filling recursive functions.

As an exercise prior to coding, students can download sample fractal programs, some quite professional, such as Fractint, from the Web and generate fractals with which to compare their own results. Hand drawing Koch curves also serves to motivate students.

Animation

The greatest obstacle to animation is student anticipation of greater effort being required. The idea of physical trajectories and producing frames/time-parametrized functions induces anxiety. To alleviate it, start with a programming exercise to draw a short horizontal line segment and then to erase the segment. Add a loop to the code to draw the segment farther to the right each time. To give an apparent smoothness of motion, have the loop not redraw the line segment but just delete one pixel on the left and add one on the right. This incremental approach "slides" the image.

The second animation exercise would be to write a function to draw a small car or, if students have learned file formats already, redraw a small scanned bitmap of a car. Combine this code with the previous nonincremental animation.

Another source of simple animations to demonstrate is the Web with its numerous animated GIF files. A frame-by-frame display of these can be accomplished by using any of several GIF-animation utilities (also on the Web). Students should be cautioned that these are often hand-drawn and are more akin to the mechanical cartoon arcades of the previous century or flip books than to computer graphics.

For nonlinear motion, a simplified solar system is an excellent programming exercise. One option is the practice in shading the planets as they move. Another optional modification is to add random periodic bumps on the surface and shaded boundaries to give the appearance of rotation and bumpiness.

At this stage, students are ready for the final project in animation. To encourage originality and imagination, several inspiring animations are demonstrated from the CDs of Bowermaster (1994) and Shatz (1993). SIGGRAPH also publishes CDs with exceptional animations but those are of higher caliber and often involve hardware and software that are not available to students.

Other Resources and Issues

Books by Clifford A. Pickover (1990, 1991) provide very readable and well-illustrated expositions on topics including computer graphics. Unlike other popularizations, Pickover's include programming code. The Graphics Gems multivolume book series also provides short expositions that could be converted into exercises or projects.

Implementation support is always a consideration, but more so in computer graphics. Without 24-bit color, these activities would suffer; other less-pressing necessities are a robust combination of memory (more than 32 MB), graphics memory (more than 4 MB), and operating system (such as UNIX, Linux, or NT).

Since students are concerned about applicability of knowledge, we have recently shifted toward OpenGL as a standard. Since NT platforms are the student preference, a second text we require for the course is Wright and Sweet (1996). If NT is not a concern, a single compact text covering both computer graphics principles and OpenGL is Angel (1997). Finally, for those not using OpenGL but using X-Windows, Pavlidis's 1982 classic has been modernized (in 1996) and expanded with exercises concentrating on interactive, event-driven programming.

References

Angel, E. (1997). *Interactive computer graphics: A top-down approach with OpenGL.* Reading, MA: Addison-Wesley.

Bowermaster, J. (1994). *Animation how-to CD.* Corte Madera, CA: Waite Group Press.

Firebaugh, M. (1993). *Computer graphics: Tools for visualization.* Dubuque, IA: Wm. C. Brown.

Hearn, D., & Baker, M. (1997). *Computer graphics* (C version). Upper Saddle River, NJ: Prentice Hall.

Lawrence, J. (1972). *A catalog of special plane curves.* New York: Dover.

Owen, G., Larrondo-Petrie, M., & Laxer, C. (1994). Computer graphics curriculum: Time for a change? *Computer Graphics, 28*(3), 183–185.

Pavlidis, T. (1996). *Interactive computer graphics in X.* Boston: Prindle, Weber, Schmidt.

Pickover, C. (1990). *Computers, pattern, chaos, and beauty.* New York: St. Martin's Press.

Pickover, C. (1991). *Computers and the imagination.* New York: St. Martin's Press.

Pokorny, C. (1994). *Computer graphics: An object-oriented approach to the art and science.* Wilsonville, OR: Franklin, Beedle, and Associates.

Schweitzer, D. (1990, February). Ray tracing: A means to motivate students in an introductory graphics course. *Proceedings of the 21st SIGCSE Technical Symposium on Computer Science Education, Washington, DC, 22*(1), 157–161.

Shatz, P. (1993). *Walkthroughs and flybys CD.* Corte Madera, CA: Waite Group Press.

Wright, R., & Sweet, M. (1996*). OpenGL superbible: The complete guide for OpenGL for Windows NT.* Corte Madera, CA: Waite Group Press.

Young, C., & Wells, D. (1994). *Ray tracing creations* (2nd ed.). Corte Madera, CA: Waite Group Press.